**CSY1018: Website Programming Assignment 2: JavaScript Assignment (50%)**

Ashleigh Claire Foster

16427265

BSc Business Computing (Systems)

University of Northampton

Submission: **Sunday, 30th April 2017, by 23h59.**

**Table of Contents**

[University of Northampton 1](#_Toc481340400)

[1. Introduction 3](#_Toc481340401)

[2. Game Design 4](#_Toc481340402)

[3. Program Design 8](#_Toc481340403)

[4. Testing 23](#_Toc481340404)

[5. Conclusion 26](#_Toc481340405)

[6. References 28](#_Toc481340406)

**Assignment brief:**

<https://nile.northampton.ac.uk/bbcswebdav/pid-3374564-dt-content-rid-3279126_1/courses/CSY1018-STD-1617/CSY1018%20Assignment%202%20-%20version%203.pdf>

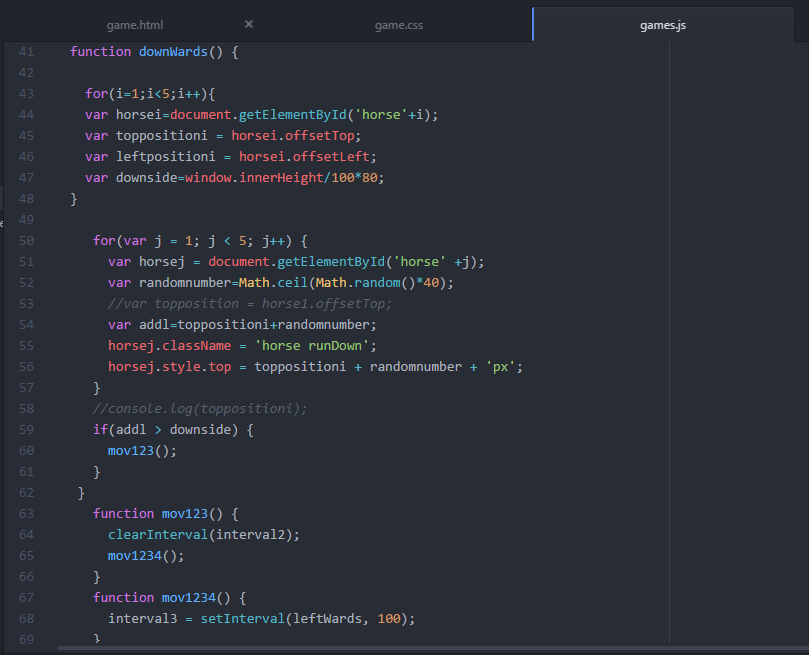
# **Introduction**

**How did you come up with a solution to meet the brief?**

The main purpose of the web-based horse racing game is that it is designed to build a horse racing game using the elements in the HTML and CSS code and then building upon those elements in the design using JavaScript to construct the movement of the horses around the track as well as applying animations to those horses depending on where the horses are in the game. For example, if the horse is running to the right of the track then the “horse runRight” function needs to be implemented, if the horse is running downwards then the “horse rundown” function needs to be implemented, if the horse is running to the left of the track then the “horse runLeft” function needs to be implemented and finally if the horse is running upwards then the “horse runUp” function needs to be applied and implemented. When the end-user presses the start button then this will allow the horses to start running around the track and then execute the relevant code afterwards. Upon further development of the game the horses could then be used as a betting game i.e. the end-user could bet on which horse would likely win the race i.e. will it be the brown horse, the blue horse, the white horse or the green horse? Depending on the set difficulty of the game depends on how many laps the horses have run i.e. if the JavaScript code has only been coded for one lap then this would only meet the first half of the solution of the assignment brief i.e. to pass the horse game with a D grade the author needs to make sure that the horses moves around the track once, then stop after the start/finish line and then are able to start back up when the start button has been pressed again. The author has also added what’s known as a “Math. Random” function in the JavaScript to counteract the different speeds that you see when the horses are going around the track. This also means that it makes it harder to identify which horse is likely to win the race as when running the code, it is usually either the brown horse, the blue horse or a tie in the horses i.e. green and blue being 1st when they stop. The horses are running/going at different speeds but they are all still turning at the exact same point around the track since all the horses have been set in a “for loop” which grabs a hold of each of the horses including the “get element by ID” code and then sets the inner height or the inner width of the horses at the same point depending on where they are on the track i.e. when the horses are running right then turn at 75% of the overall screens size, when the horses are turning to go to the left then it is covering 70% of the overall screen size, when the horses are about to run up then it is covering 10% of the overall screen size and the same applies for when the horses are about to turn right on the track i.e. 10% of the overall screens size.

# **Game Design**

**What did you need to do to work out how to get the horses to stop/turn at the relevant points on the track?**

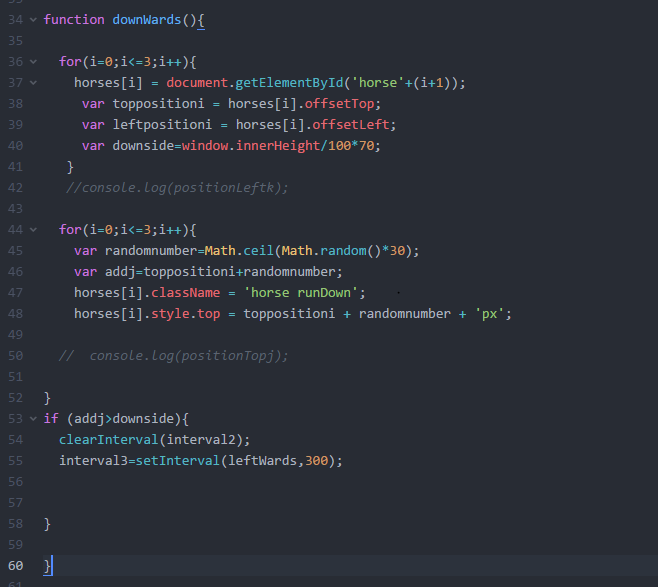
The author needed to make sure that to turn the horses at the relevant point on the track that they must set the inner width and the inner height for each side of the track so that it could turn at the related and relevant point. The author did this by setting both the right-hand side and the downside of the track to an inner height and inner width of dividing by 100 and multiplying by 80 making the horse cover 80% of the screen i.e. for the right-hand side it would be ***window.innerWidth/100\*80*** and for going down it would be ***window.innerHeight/100\*80***. For the left-hand and for going upwards the author again set the inner width and the inner height for each of the sides of the track so that it could turn at the related and relevant point but this time the author had to set the inner height and inner width of the track to divide by 100 and multiply by 5. For example, for the horses going to the left-side of the track it was set to ***window.innerWidth/100\*5*** and for going upwards on the track it was set to ***window.innerHeight/100\*5***. The author had to make sure that for each of the turning functions that it was set to its appropriate animation i.e. for running right it would be ***‘horse runRight’***, for running down it would be ***‘horse runDown’***, for running to the left it would be ***‘horse runLeft’*** and for running upwards it would be ***‘horse runUp’***. These animations were already provided in the CSS file so the author just had to pick these CSS classes out of the “game.css” file. The author had to set the same “for loop” for each of the individual turns twice that were set on the horses as the author wanted to make sure that each of the horses turned rather than just setting it just to one specific horse i.e. to horse1. The for loop basically describes that if there are less than 5 horses but equal to one horse then get a hold of all 4 horses and increment it by 1 which is the ++ part of the for loop i.e. ***for(i=1;i<5;i++)*** is the for loop for setting the top and left position as well as setting the turns for each of the individual horses and the ***for(var j = 1; j < 5; j++)*** for loop is setting up the relevant functions i.e. runRight as well as making sure that each individual pixel is set for the movement around the track. Here is an example of the downwards function implementing the “turns” in the relevant points of the track:

Setting the relevant functions so that each of the horses in the game can run by a certain pixel ‘px’ and produce an animation i.e. ‘horse runDown’ as well as adding a math random number which is the next part of this report which allows the horses to take over each other.

Setting each of the individual functions so that it firstly clears each of the intervals and then sets them i.e. clearing the previous turn function before producing another turn function.

Setting the turns for each of the individual horses using the innerHeight method to calculated 80% of the screens height.

Setting how fast or how slow the horse runs in each turn interval. In this case 100 milliseconds per pixel.



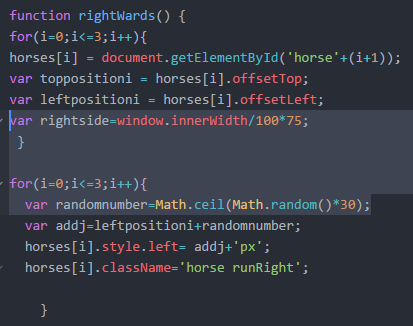
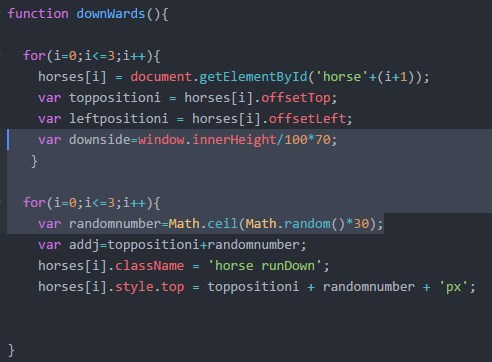
The author has then made the following improvements to the overall design of the game so that the horses turned not only at a relevant point in the track but also to prevent them from bunching up together as much as they were doing originally. The author has changed the heights and widths of each of the turn functions as well as changing the intervals slightly to reduce repetitive unnecessary code. The “for loops” and the horse’s names have also been changed. This is because the author has now included an array list of horses rather than grabbing all the horses in the for loop as this will allow more control over the horses turning points on the track. The same concept applies as above though.

**How did you design the game so that a different horse won each time?**

The author has added what is known as a “Math. Random” function in the JavaScript to counteract the different speeds that you see when the horses are running and going around the track. This means that it’ll make it harder to identify which horse is likely to win the race as when running the code, it is usually either the brown horse, the blue horse or a tie between the horses i.e. green and blue being 1st when they stop or blue and brown being 1st etc. There isn’t really a specific interval set which has been inputted into the JavaScript code which allows one of the horses to overtake another horse on the track. The intervals have purely been set to allow the horses to go around the track as soon as the start button is pressed rather than having a different horse win the game each time. Here are a few examples of where the author has used the “Math.Random” function to change the speed of each of the horses as soon as the start button has been pressed:

The functions which are highlighted in the blue boxes are the functions used to change the speed of the horses.

**How did you design the game to make it interesting? E.g. so overtaking happens**

The author hasn’t really specified and or implemented exactly how the horses are overtaking in the horse game, however, the author has used the “*Math.Random*” function to allow the horses to move in various positions along the track whilst they are running along each side of the track. The speeds at which each of the horses are running could also be a factor in how the overtaking process happens within the game. For example, the slower the horses are running in the game the less likely they are to overtake each other which is similar to when the horses run right and then down on the track whereas when the horses are running at a faster speed then they are more likely to overtake each other just like how it is currently set up for when the horses run left and upwards in the java game. The higher the Math.Random integer the more likely the horses are going to spread apart and overtake each other whilst they are running along the track whereas the lower the Math.Random integer the less likely the horses are going to spread apart and or overtake each other.

Here are a few examples of how setting the inner height and the inner width of the overall size of the screen as well as setting the Math.Random random allows the horses to overtake each other.

# **Program Design**

**How did you use language tools to reduce repeated code?**

The languages tools which the author has used to reduce repetitive code is the horse’s arrays and the for loops as well as declaring the different horse’s intervals at the top of the JavaScript code rather than just copying and pasting the same code each time the interval needs changing in an if statement. The reason why the author has used arrays for each of the horses is because otherwise every time you are creating a function whether it be for turning the horses around the track, stopping the horses or starting the horses then it reduces the code by at least 5 lines. For example, in an array, the author has included a blank array value “*var horses = []*” which is basically stating that however many horses there are in the JavaScript horse game then get a hold of those horses and perform the necessary requirements. If the array hadn’t been used then it would look something like this:

*var horses = [];*

*horses[0];*

*horses[1];*

*horses[2];*

*horses[3];*

Regarding the “for loop” in the JavaScript horses game. The “for loop” has been implemented into the JavaScript code so that the author and or web browser can get a hold of each of the different horses within the game from 0 – 3 and then use the [i] to increment the array of horses which has been declared at the top of the horse game. The for loop also allows you to run the same JavaScript code in a continuous loop each time just with a different interval or integer set to it. If the author hadn’t implemented a “for loop” in the code then they would constantly be repeating the same block of code repeatedly whilst wasting a lot of valuable time. If the for loop hasn’t been used then the code would look something like this:

*var horses = document.getElementById(horses1)* ;

*var topposition = horses1.offsetTop;*

*var leftposition = horses1.offsetLeft;*

*var rightside=window.innerWidth/100\*80;*

*var randomnumber=Math.ceil(Math.random()\*40);*

*var addj=leftpositioni+randomnumber;*

*horses1.style.left= addj+'px';*

*horses1.className='horse runRight';*

*var horses = document.getElementById(horses2)* ;

*var topposition = horses2.offsetTop;*

*var leftposition = horses2.offsetLeft;*

*var rightside=window.innerWidth/100\*80;*

*var randomnumber=Math.ceil(Math.random()\*40);*

*var addj=leftpositioni+randomnumber;*

*horses2.style.left= addj+'px';*

*horses2.className='horse runRight';*

*var horses = document.getElementById(horses3)* ;

*var topposition = horses3.offsetTop;*

*var leftposition = horses3.offsetLeft;*

*var rightside=window.innerWidth/100\*80;*

*var randomnumber=Math.ceil(Math.random()\*40);*

*var addj=leftpositioni+randomnumber;*

*horses3.style.left= addj+'px';*

*horses3.className='horse runRight';*

*var horses = document.getElementById(horses4)* ;

*var topposition = horses4.offsetTop;*

*var leftposition = horses4.offsetLeft;*

*var rightside=window.innerWidth/100\*80;*

*var randomnumber=Math.ceil(Math.random()\*40);*

*var addj=leftpositioni+randomnumber;*

*horses4.style.left= addj+'px';*

*horses4.className='horse runRight';*

Whereas a whole bunch of that code could be adjusted to just a few lines like this:

*for(i=0;i<=3;i++){*

*horses[i] = document.getElementById('horse'+(i+1));*

*var toppositioni = horses[i].offsetTop;*

*var leftpositioni = horses[i].offsetLeft;*

*var downside=window.innerHeight/100\*70;*

*}*

*for(i=0;i<=3;i++){*

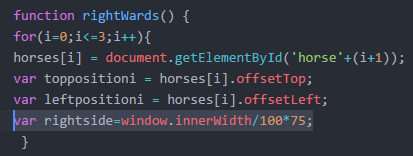
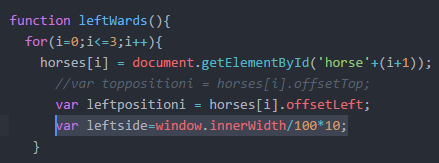
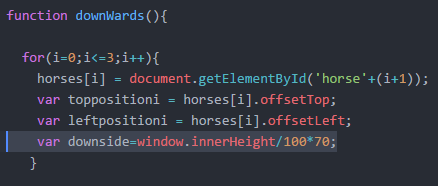
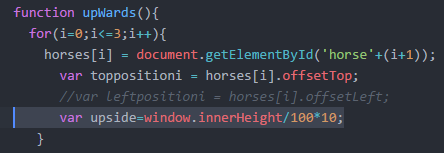
*var randomnumber=Math.ceil(Math.random()\*30);*

*var addj=toppositioni+randomnumber;*

*horses[i].className = 'horse runDown';*

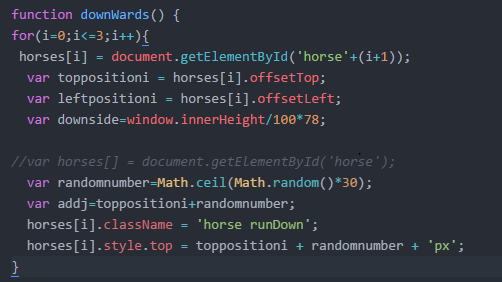
*horses[i].style.top = toppositioni + randomnumber + 'px';*

**What did you have to do to account for different screen sizes?**

To account for the different screen sizes within the horse's JavaScript game the author had to declare either the inner width or the inner height of the overall screen size. For example, in the rightwards function for when the babies are turning right, the horses would be turning at around about 75% of the screens inner width with both the top position and the left position set for each of the horses whilst going around the track. For the leftwards function for when the babies are turning left the horses would be turning at around about 10% of the screens inner width with both the top position and the left position set for each of the horses whilst going around the track. For the downwards function for when the babies are turning down the horses would be turning at around about 70% of the screens inner height with both the top position and the left position set for each of the horses whilst going around the track. For the upwards function for when the babies are turning upwards, the horses would be turning at around about 10% of the screens inner height with the top position being set for each of the horses whilst going around the track. Here are a few examples in my JavaScript code of where I have included and or accounted for the different screen sizes:

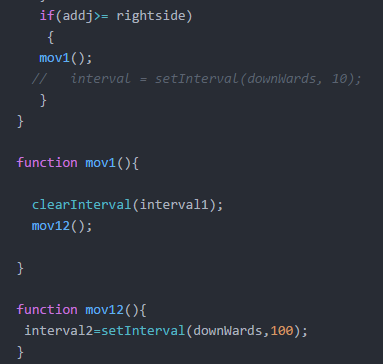
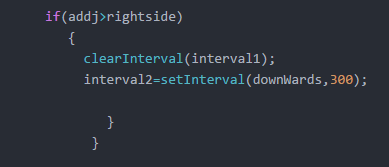
Here is an example of the inner height and inner width being set for when each of the 4 horses are turning around the track. As shown in the screenshot examples, the right side of the screen has been set to 75% of the inner width, the downside has been set to 70% of the inner height, the left side has been set to 10% of the inner width and finally the upside i.e. the horses going up has been set to 10% of the inner height of the window.

**Did you consider any alternative approaches? If so, why did you choose the approach you decided to use?**

Yes, the author did consider a variety of other ways and or solutions in how to solve this problem. The first way in which the author decided to solve this problem was by calling all the horses by the name “horsei” and then setting that horsei into different for loops to control the horses. The only problem with this method was that it made the horses completely bunched up together i.e. they wouldn’t separate apart at all which made it complicated when testing the application out as you couldn’t really see where each of the four horses were, only slightly when they moved positions on Math.Random.

**Before**

**After**

Another alternative approach which the author has originally thought of was the idea of adding various functions to the JavaScript horse game but these functions proved to be problematic and came with their flaws i.e. the author couldn’t stop the horses properly or restart the “start” button and the various functions and intervals sometimes made the horses come off the track and go in different directions. Therefore, the author has decided it was best to completely remove those different function names and make the intervals as small “if statements” instead. Here is an example of how the author originally planned it on the left-hand side and how it is currently set out now on the right-hand side:

**After**

**Before**

**How did you use the features of JavaScript to do this?**

* Conditional statements
* Functions
* Intervals

**Conditional statements**

Conditional statements are a part of JavaScript and other programming languages like Java and C etc. A conditional statement is used when a certain action needs to be performed a certain way. For example, the “if statements” usually refer to a specific piece of code which is “true” whereas an else statement usually refers to a specific piece of code which is “false”. In assignment 2 the author has used the following conditional statements to control the movement of the horses around the track:

if(addj>rightside)

{

clearInterval(interval1);

interval2=setInterval(downWards,300);

}

if (addj>downside){

clearInterval(interval2);

interval3=setInterval(leftWards,300);

}

if (addj<leftside){

clearInterval(interval3);

interval4=setInterval(upWards,300);

}

if(addj<upside){

clearInterval(interval4);

// a++;

interval5=setInterval(stopHorses,300);

}

if(addj>rightside)

{

if(a>1) {

}

else{

a++;

clearInterval(interval5);

startFunction();

a = 0;

}

}

if(a<=1){

interval1=setInterval(rightWards,300);

}

else{

for(i=0;i<=3;i++){

horses[i] = document.getElementById('horse'+(i+1));

horses[i].className= 'horse standRight';

}

}

**Functions**

The main purpose of using a function as a feature of JavaScript is that it splits certain parts of the code into a whole block of code and depending on where that function is placed in the JavaScript file depends on which function is going to get executed first and which function is going to get executed last. The code that is inside a function is executed depending on what has been written in that function. In the case of the horse JavaScript game, the following code below requires functions so that the horses can move and animate accordingly around the track otherwise the game wouldn’t work at all without implementing functions, especially as it relies heavily on this information (which of course is all the code except the load event and the var’s which are declared at the top):

function rightWards() {

for(i=0;i<=3;i++){

horses[i] = document.getElementById('horse'+(i+1));

var toppositioni = horses[i].offsetTop;

var leftpositioni = horses[i].offsetLeft;

var rightside=window.innerWidth/100\*75;

}

for(i=0;i<=3;i++){

var randomnumber=Math.ceil(Math.random()\*30);

var addj=leftpositioni+randomnumber;

horses[i].style.left= addj+'px';

horses[i].className='horse runRight';

}

if(addj>rightside)

{

clearInterval(interval1);

interval2=setInterval(downWards,300);

}

}

function downWards(){

for(i=0;i<=3;i++){

horses[i] = document.getElementById('horse'+(i+1));

var toppositioni = horses[i].offsetTop;

var leftpositioni = horses[i].offsetLeft;

var downside=window.innerHeight/100\*70;

}

for(i=0;i<=3;i++){

var randomnumber=Math.ceil(Math.random()\*30);

var addj=toppositioni+randomnumber;

horses[i].className = 'horse runDown';

horses[i].style.top = toppositioni + randomnumber + 'px';

}

//console.log(toppositioni);

if (addj>downside){

clearInterval(interval2);

interval3=setInterval(leftWards,300);

}

}

function leftWards(){

for(i=0;i<=3;i++){

horses[i] = document.getElementById('horse'+(i+1));

//var toppositioni = horses[i].offsetTop;

var leftpositioni = horses[i].offsetLeft;

var leftside=window.innerWidth/100\*10;

}

for(i=0;i<=3;i++){

var randomnumber=Math.ceil(Math.random()\*90);

var addj =leftpositioni-randomnumber;

horses[i].style.left= addj+'px';

horses[i].className='horse runLeft';

}

if (addj<leftside){

clearInterval(interval3);

interval4=setInterval(upWards,300);

}

}

function upWards(){

for(i=0;i<=3;i++){

horses[i] = document.getElementById('horse'+(i+1));

var toppositioni = horses[i].offsetTop;

//var leftpositioni = horses[i].offsetLeft;

var upside=window.innerHeight/100\*10;

}

for(i=0;i<=3;i++){

var randomnumber=Math.ceil(Math.random()\*30);

var addj=toppositioni -randomnumber;

horses[i].style.top= addj+'px';

horses[i].className = 'horse runUp';

}

if(addj<upside){

clearInterval(interval4);

// a++;

interval5=setInterval(stopHorses,300);

}

}

function stopHorses(){

for(i=0;i<=3;i++){

horses[i] = document.getElementById('horse'+(i+1));

var toppositioni = horses[i].offsetTop;

var leftpositioni = horses[i].offsetLeft;

var rightside=window.innerWidth/100\*30;

}

for(i=0;i<=3;i++){

var randomnumber=Math.ceil(Math.random()\*90);

var addj=leftpositioni+randomnumber;

horses[i].style.left= addj+'px';

horses[i].className='horse runRight';

}

if(addj>rightside) //Conditional statement inside the brackets "addj>rightside".

{

if(a>1) {

}

else{

a++;

clearInterval(interval5);

startFunction();

a = 0;

}

}

}

function startFunction(){

if(a<=1){

interval1=setInterval(rightWards,300);

}

else{

for(i=0;i<=3;i++){

horses[i] = document.getElementById('horse'+(i+1));

horses[i].className= 'horse standRight';

}

}

}

function myLoadEvent(){

var start= document.getElementById('start');

start.addEventListener('click', startFunction);

var startline = document.getElementById('startline');

var topstart = startline.offsetTop;

var leftstart = startline.offsetLeft;

}

**Intervals**

The main purpose of using an interval as a feature of JavaScript is that it helps control the time of each of the elements within the website such as different objects like horses or when you want to time a certain event such as a pop-up message box on a website coming up after 6 seconds (6000). In the case of the horse JavaScript game, the intervals are being set to control how fast or how slow the horses are going around the track. When the horses are running right and running down they are running a lot slower than they are when the horses are running to the left or upwards. This is because of the Math.Random function and the set 300 configuring with each other to change the speed and intervals at which the horses in the game run around the track. Clearing the intervals in the code makes sure that intervals aren’t overlapping each other otherwise the rest of the intervals simply wouldn’t work and or the horses won’t run at all due to the slight error. Here are a few examples of where the author has used intervals in the horse game:

if(addj>rightside)

{

clearInterval(interval1);

interval2=setInterval(downWards,300);

}

if (addj>downside){

clearInterval(interval2);

interval3=setInterval(leftWards,300);

}

if (addj<leftside){

clearInterval(interval3);

interval4=setInterval(upWards,300);

}

if(addj<upside){

clearInterval(interval4);

// a++;

interval5=setInterval(stopHorses,300);

}

if(addj>rightside) //Conditional statement inside the brackets "addj>rightside".

{

if(a>1) {

}

else{

a++;

clearInterval(interval5);

startFunction();

a = 0;

}

}

function startFunction(){

if(a<=1){

interval1=setInterval(rightWards,300);

}

else{

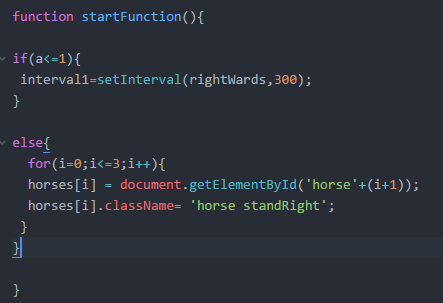
for(i=0;i<=3;i++){

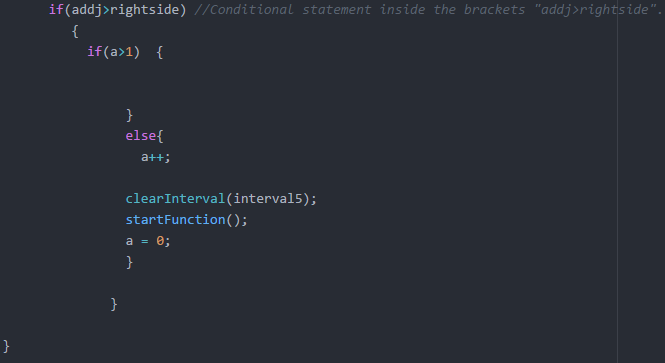
horses[i] = document.getElementById('horse'+(i+1));

horses[i].className= 'horse standRight';

}

}

}



**How did you break the problem up into different tasks?**

The first approach the author took to break the problem down into different tasks was to firstly consider which was the best approach to firstly implement the overall structure of the JavaScript. The author decided that it was best to start off with “Topic 3” to get an overall sense of where elements needed to be placed in the JavaScript and to get a rough idea on how to break this problem into different sections. The JavaScript code which originated from this lesson was the offset top and offset left configurations in each of the functions, the Math.Random and the way the inner height and the inner width were constructed. This then allowed the author to split up a similar code for each of the individual functions i.e. rightwards, downwards, leftwards, upwards and stop horses depending on where the horses were in the game. Then add the “document.addEventListener(‘DOMContentLoaded’, myLoadEvent” to the end of the code. The load event part of the JavaScript code was also taken from the topic 3 as an idea and was built upon.

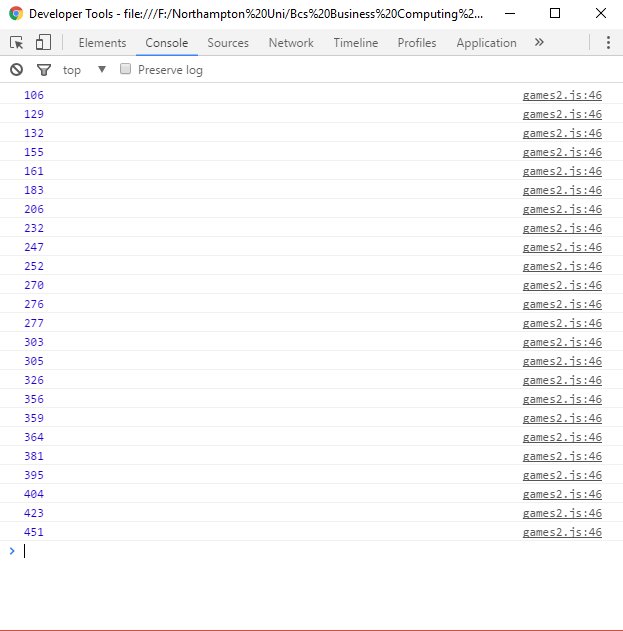
After this, the author then decided to use arrays to help structure the horses out rather than having to repeat the same code for each of the individual horses. Based on this assumption the author had come up with the idea of using arrays and for loops to control the horses as they move around the track and execute the functions. The arrays can be seen in the for loop as representing the letter “i” whilst controlling the 4 horses in that for loop i.e. 0, 1, 2 and 3. The author needed to make sure that the array's variable was declared at the top using “var horses []” as the square brackets indicate a blank array is taking place meaning any integer (or a number of horses) could be passed through that array. Each element then had to refer to “horses[i]” as the common basis for controlling each of the horses together.

The author then finally came up with a set of intervals which needed to be implemented into the JavaScript code so that the horses could change or vary in speed whilst going around the track depending upon the horse’s current position on the track and the way in which the Math.Random variable was set up for each of the 4 main functions of the JavaScript game. Each of these intervals needed to be set up in “if statements” so that the speed of the horses could be controlled as well as cleared each time a new interval was set.

# **Testing**

**How did you test that your code worked?**

To test that the code worked effectively and efficiently the author had to make sure that they had used the “developer tools” in Google Chrome so that any JavaScript errors which popped up were fixed. The JavaScript errors were usually presented to the author in a red text. The usual errors which popped up were either a missing bracket in the JavaScript .js code or a missing variable. The console in Google Chrome helped the author detect which line/lines were having the problem and or issue and how to fix it. The author has also opened the .html file in the Google Chrome browser to check whether the game was working correctly as it should do or not. Another way the author tested to make sure that their code worked was through the console.log command which was typed into the JavaScript to check for any errors or response to certain functions. For example, if I wanted to check whether the left position command was working the author would use “console.log(leftpositioni-1)” to find out whether there are any errors in the code and to see if there are any of the numbers showing up in blue in the console log. Here is an example of how the developer tools console works when testing the horses game solution:



Here is an example of the console log (console.log). As shown in this console log, when the horses are moving down it will detect the top position of each of them and then shows the exact position of those horses through a series of numbers. Line 46 is basically the line where the “console.log(toppositioni);” has been added to detect the top position for each of the 4 horses. This console is an inbuilt feature in the Google chrome browser and doesn’t need downloading.

**Could you test certain aspects of the code without running the entire game and waiting for it to finish?**

You could test the code without running the entire game and waiting for it to finish by at least running half of the game. This could include making sure that the horses are moving at the start of the game to the right-hand side of the track and then turning to move downwards on the track. This would then include half of the track being used meaning that half of the horse’s game would have run rather than the full horses game. But to make sure that the game has been properly tested the end-user would have to run the entire game including waiting for it to finish at the start/stop line if the end-user did want to test the horses game as not all the tests would apply when running the game for just half of the overall track.

**What tests did you carry out and what were the outcomes?**

|  |  |  |  |
| --- | --- | --- | --- |
| What are you testing? | Expected outcome | Actual outcome | Any improvements? |
| Each horse should move to the top right corner of the track, then down the right-hand side of the track, along the bottom and back up to the top turning whilst ensuring they stay on the track. | The 4 horses should turn around each side of the track i.e. right, down, left and up without any problems. | Yes, the 4 horses turn exactly as they should do around the track without any problems. | Some improvements the author could make when the horses are turning around the track is to unbunch the horses by passing arguments, a speed function and setting intervals for each of the horses as they turn. |
| The horses should continue following the track until they reach the start/finish line. | The horses should continue following the track until they reach the start/finish line. | Yes, the 4 horses do follow the whole track until they reach the start/finish line. | Some improvements the author could make when the horses continue to follow the track is to make sure that the horses don’t go off the side of the track on the odd occasion when they are running around the track. |
| As horses change direction and start/stop their animations should change. The animations are provided for you as CSS classes. The horses should always face the direction they are traveling. | The horses should change their animations and be facing that general direction when running that side of the track. The CSS classes should be provided in the JavaScript code too. | Yes, the 4 horses do in fact change animation when they are running around the track as well as facing that general direction of travel. The CSS classes have also been used such as “horse runRight”. | Nothing needs changing. No improvements need to be made. |
| As each horse reaches the start/finish line it completes the lap and should stop racing. The horses don’t have to stop exactly on the line but must go over it. | The 4 horses should reach the finishing line and stop just after that finishing line without any problems on the first lap of the game. | Yes, the 4 horses do in fact go around the track and stop after the finishing line without any problems on their first lap around the track. | The author could improve on this by making sure that other laps after the first lap stop the horses as well rather than just the once. |
| Pressing the start button again should reset the horses and start another lap. | When the end-user presses the start button again the horses should start racing again around the track. | Yes, this did work. When the end-user did press the start button again all four horses fired up and started running around the track again. | None. No improvements need to be made for this section. |

**What bugs did you discover during testing?**

There were a series of bugs which popped up during the testing stages of the application. The first bug which popped up whilst testing the JavaScript horse game was the “start button” itself. When the end-user accidently presses the start button twice the horses run a lot faster than they were set up to do. This then causes the horses to move sporadically across the screen i.e. when the horses are turning and about to move downwards the horses do indeed move down but halfway down the right-hand side of the track whilst they are going down they suddenly drift off the screen. Another bug (although it could be fixed by completely changing the code) is the horses being squashed together whilst they are running together. This is because the JavaScript code is getting a hold of each of the 4 horses and then setting them all under one interval rather than setting an interval or arguments for each of the individual horses.

# **Conclusion**

**What works well?**

One aspect which works quite well in the JavaScript horse game is the animations and the direction in which the horses are traveling/running at. For example, when the horses are running to the right the “horses runRight” animation will be implemented and all the horses will be facing right as they are running along the screen. There aren’t any glitches at all in this part of the code as it is straight forward and self-explanatory. Another feature which works quite nicely in the horse game is the horses starting another lap as soon as the end-user presses the “Start” button in the horse game as there aren’t any glitches in this process (unless you press the start button twice whilst the horses are running).

**What improvements could/can be made?**

*\*\*Refer to the test plan for a list of improvements for each of the tested scenarios that needed to be made to the JavaScript horse application.*

Some improvements the author could make when the horses are turning around the track is to unbunch the horses by passing arguments, a speed function and setting intervals for each of the horses as they turn.

The author could make when the horses continue to follow the track is to make sure that the horses don’t go off the side of the track on the odd occasion when they are running around the track.

The author could improve on this by making sure that other laps after the first lap stop the horses as well rather than just the once.

**What else would you have done if you had more time?**

If the author had more time to implement the JavaScript game the author would have changed the JavaScript coding around slightly different by adding more intervals as well as passing more arguments and functions so that the horses would run around the track without being jumbled together. The author would have tried to make sure that using those intervals and arguments not only unjumbled the horses but also allowed the horses to operate at different speeds around the track i.e. one horse running slower than the other horse and or overtaking. The author could have made sure that the horses didn’t turn at the exact same point in the track as this would prevent the horses from overlapping each other whilst they are turning around each side of the track. The author could have also used one of the topics in the lesson to detect which horse will likely win the race once they have all crossed the finish line. The name of the horse should then be displayed in the form of a “pop-up” prompt which informs the end-user that horse 1 won or horse 4 lost etc. The author could have disabled the start button whilst the race is in progress so that the horses don’t speed up and go off to the side when the start button accidently gets pressed twice.

**If you had to build a similar game in the future, what would you do differently and why?**

If the author had to make or build a similar game in the future the author would have added more features to the game such as collision detection objects i.e. when the horse is running around the track it would detect a hurdle and then jump over that hurdle before carrying on and running the rest of the track. Another way in which the author could build a similar game in the future is by simply planning it out a lot better and gather more knowledge from external sources to help with the overall construction of the game rather than relying on limited knowledge. The author could form a pseudocode or flowchart like a plan to make sure that they plan the design first before making any implementations to the game.

# **References**

Javascript If...Else Statements". https://www.w3schools.com/js/js\_if\_else.asp. N.p., 2017. Web. 26 Apr. 2017.

CSY1018 - Web Programming". https://nile.northampton.ac.uk/bbcswebdav/pid-3374564-dt-content-rid-3279126\_1/courses/CSY1018-STD-1617/CSY1018%20Assignment%202%20-%20version%203.pdf. Moulton College. N.p., 2017. Web. 5 Feb. 2017.

https://nile.northampton.ac.uk/webapps/blackboard/content/listContent.jsp?course\_id=\_64602\_1&content\_id=\_3087458\_1. Tom Butler. N.p., 2017. Web. 5 Feb. 2017.